Computers & Geosciences 70 (2014) 110–119

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

A Map-Reduce-enabled SOLAP cube for large-scale remotely sensed data aggregation

Jiyuan Li [a,b,*], Lingkui Meng [a], Frank Z. Wang [b], Wen Zhang [a], Yang Cai [c]

[a] School of Remote Sensing and Information Engineering, Wuhan University, Wuhan 430079, China
[b] Future Computing Group, School of Computing, University of Kent, Kent ME44AG, UK
[c] Water Resources Information Centre, MWR, Beijing 100053, China

ARTICLE INFO

Article history:
Received 13 March 2013
Received in revised form
30 March 2014
Accepted 19 May 2014
Available online 29 May 2014

Keywords:
SOLAP
Spatio-temporal cube
Data-intensive computing
CyberGIS
Environment monitoring

ABSTRACT

Spatial On-Line Analytical Processing (SOLAP) is a powerful decision support systems tool for exploring the multidimensional perspective of spatial data. In recent years, remotely sensed data have been integrated into SOLAP cubes, and this improvement has advantages in spatio-temporal analysis for environment monitoring. However, the performance of aggregations in SOLAP still faces a considerable challenge from the large-scale dataset generated by Earth observation. From the perspective of data parallelism, a tile-based SOLAP cube model, the so-called Tile Cube, is presented in this paper. The novel model implements Roll-Up/Drill-Across operations in the SOLAP environment based on Map-Reduce, a popular data-intensive computing paradigm, and improves the throughput and scalability of raster aggregation. Therefore, the long time-series, wide-range and multi-view analysis of remotely sensed data can be processed in a short time. The Tile Cube prototype was built on Hadoop/Hbase, and drought monitoring is used as an example to illustrate the aggregations in the model. The performance testing indicated the model can be scaled along with both the data growth and node growth. It is applicable and natural to integrate the SOLAP cube with Map-Reduce. Factors that influence the performance are also discussed, and the balance of them will be considered in future works to make full use of data locality for model optimisation.

1. Introduction

Fields describes physical phenomena that changes continuously in time and/or space (Vaisman and Zimányi, 2009). Raster is the most popular field data model, and other examples include Voronoi diagrams and Triangulated Irregular Network (TIN). As the typical data structure of remotely sensed data, raster divides the world into a regular grid of cells and assigns attributes to the cells (Longley et al., 2001). In recent years, some efforts have been made to integrate field data into spatial On-Line Analytical Processing (OLAP) models, a paradigm aimed at exploring spatial data by drilling on maps, in the same way as OLAP operates over tables and charts (Bédard et al., 2007; Ahmed and Miquel, 2005; Bimonte et al., 2010a; Gómez et al., 2010a). Those works regard the discrete field data as measures and employ Map Algebra (MA) /Multi-dimensional Map Algebra (MMA) to achieve the spatio-temporal aggregations. MA (Tomlin, 1991) is a powerful language tool for spatial modelling and can be classified as Local, Focal and Zonal functions. MMA extends the conventional map algebra and enables it to handle spatio-temporal situations (Mennis et al., 2005). Mennis also indicates that large-scale data puts heavy pressure on the efficiency of MMA. The aggregations are the basic computation of SOLAP (Spatial OLAP) when performing multi-dimensional analysis (Zhang, 2006; Lopez et al., 2005), and thus, aggregation performance directly affects the efficiency of SOLAP.

On the other hand, the incorporation of the latest-generation sensors for Earth observation has produced the data-intensiveness problems (Lee et al., 2011) that exert heavy pressure on the efficiency of SOLAP, especially for spatio-temporal aggregation. Apart from optimised MA/MMA algorithms (Mennis, 2010), using the technologies of distributed/parallel/GPU-accelerated computing to improve MA/MMA is also a potential way to accelerate the aggregation (Shrestha et al., 2006; Zhang et al., 2010; Xie et al., 2012). However, the scalability and fault tolerance of the computing paradigm these works are based on still face challenges in regard to large-scale data.

Map-Reduce (M-R), proposed by Google, is a data-intensive computing paradigm that combines the ideas of data locality and data parallelism (Dean and Ghemawat, 2004) for massive data. The systems based on this paradigm can be scaled to thousands of

* Corresponding author at: School of Remote Sensing and Information Engineering, Wuhan University, Wuhan 430079, China. Tel.: +86 18672375381.
E-mail address: homegis.lee@gmail.com (J. Li).

http://dx.doi.org/10.1016/j.cageo.2014.05.008
0098-3004/

commodity machines (White, 2009). Considering the advantages of M-R, combining OLAP and this paradigm has led to a new research topic for big data analysis (Nandi et al., 2011; Abelló et al., 2011). To benefit from the M-R computing paradigm also in the context of SOLAP for raster, a tile-based SOLAP cube model, the so-called Tile Cube, is proposed in this paper. The main contribution of Tile Cube is an M-R approach to perform Roll-Up/Drill-Across operations in the SOLAP environment to improve the throughput and scalability of raster aggregation. The model is implemented on top of Hadoop (http://hadoop.apache.org) and HBase (http://hbase.apache.org), and the complexity of the distribution is encapsulated by the elegant and simple interfaces.

The reminder of this paper is organised as follows. Section 2 recalls the fundamentals of raster SOLAP and M-R. Our model concept and different aggregations in the model are proposed in Section 3. Section 4 presents the Tile Cube implementation in the Hadoop/HBase environment and Section 5 describes the performance testing. Then, related works are presented in Section 6. Finally, Section 7 concludes with a discussion of on-going research.

2. Background

2.1. SOLAP cube

The typical spatial OLAP cube can be generally described as a tuple (D, M, F), where (1) D is a set of *dimensions*, and the *dimension schema* can be described as a tuple $(L, \rightarrow)$, where L is a finite set of *levels* $\{l_1, \ldots, l_j\}$, and each level $l \in L$ includes several *members*. "$\rightarrow$" refers to the aggregation relation between two levels (e.g., $l_i \rightarrow l_j$ denotes the data in l_i is aggregated to l_j). D includes *spatial* and *non-spatial dimensions*. Each member in levels of spatial dimension is a geographic object. (2) M is a set of *measures*. A measure describes the attribute value of an object determined by dimensions. Two commonly used measures are: *numeric measure* and *spatial measure*. A *fact* is a set of dimension members and measure values. (3) F is a set of aggregation functions that refer to the calculation methods for the *fact*.

A SOLAP cube is an interactive model that highlights the multi-view information discovery. Various OLAP operations, including *Dice*, *Slice*, *Roll-Up*, *Drill-Across* and *Pivoting*, can be manipulated on the cube. However, this model is designed for discrete spatial data. Gómez et al. perceived a field as a typical SOLAP cube and used the term "tessellation" to discretise the field (Gómez et al., 2010b). Bimonte introduced the field object measure and field hierarchies under the support of geographic objects (Bimonte and Myoung-Ah, 2010b). In particular, McHugh extended spatial dimension to "matrix (raster) dimension" and presented the concept of "matrix cube" where each fact is associated with a raster cell and its attributes (McHugh, 2008). The aggregations in field/raster SOLAP are usually implemented by MA/MMA, as shown in Fig. 1.

2.2. Map-Reduce paradigm

Map-Reduce achieves large-scale data parallel computation using Map and Reduce functions. As shown in Fig. 2, each M-R Job (*an entire batch processing*) takes a Key-Value (K-V) dataset as input: (1) the Map phase executes user functions and transforms the input K-V pairs into intermediate K-V pairs: $[\langle k1, v1\rangle] \rightarrow [\langle k2, v2\rangle]$. (2) In the Shuffle phase, the intermediate results are grouped by key and then sent to the nodes performing the Reduce function: $[\langle k2, v2\rangle] \rightarrow \langle k2, [v2]\rangle$. (3) The Reduce phase calls user functions to process the results from Maps and outputs the final results: $\langle k2, [v2]\rangle \rightarrow \langle k3, v3\rangle$. Multiple keys can be used to operate multidimensional dataset, and complex processes can be implemented by chaining multiple M-R jobs.

In a distributed environment, the raster dataset, having natural data parallelism, is usually partitioned into multi-blocks by the same boundary size to achieve the balanced load of storage/computation. Therefore the partitioned raster dataset can be regarded as a K-V dataset by identifying the key of each block as its dimension information. In view of this consideration, the architecture of a Tile Cube including three layers is described in Fig. 3. In the application layer, multi-source spatial information is mapped to the cube model, which enables analysts to operate interactively. The physical layer refers to an M-R environment where each compute node is a data node (*called Data/Compute Node*, DCN). In this environment, the raster tile is regarded as a basic spatial measure, and the aggregations in/between the cubes are decomposed into multiple distributed map algebra functions in the M-R pipeline. We focus on the working mechanism in the logical layer and answer how to transform the aggregations of the cube to M-R Jobs.

3. Tile Cube model

3.1. Concept of Tile Cube model

In Tile Cube, the time-series raster is perceived as a typical data cube. The global subdivision grid, which partitions the continuous geographical surfaces into seamless and multi-level cells, is employed to partition the cube into massive tiles. Fig. 4(a) shows a global logical tile scheme based on a latitude/longitude grid (Sample and Loup, 2010). This scheme partitions the surface by the $2^{n+1} \times 2^n$ method, and thus, the first layer is a 2(*longitude*) $\times$ 1 (*latitude*) grid, and the finer grid can be generated by quad-tree partitioning on the first one. Based on this partition, the model is distinguished by the following features.

3.1.1. Spatial measure

The partitioned cube maps data values to regularly tiled planimetric and temporal positions. The ***Tile*** is regarded as the basic spatial measure as well as storage/computation unit.

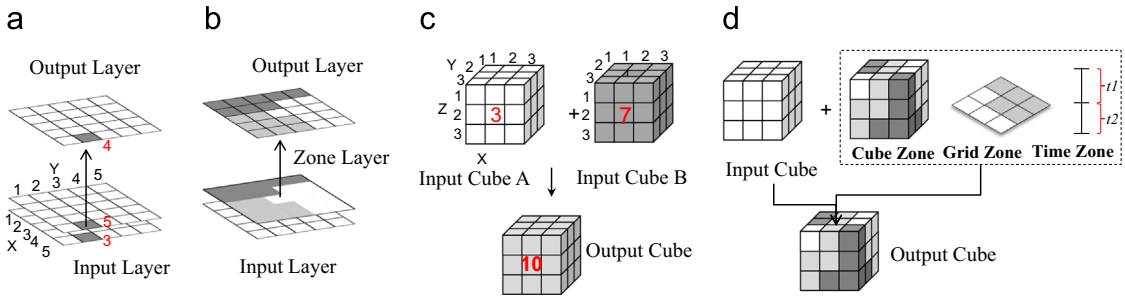

Fig. 1. Schematic diagram of map algebra and multidimensional map algebra.

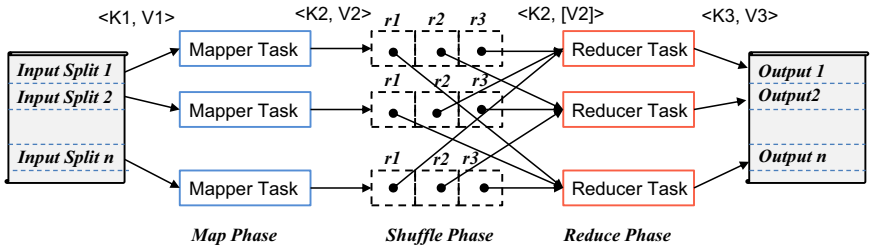

Fig. 2. Schematic diagram of map-reduce paradigm.

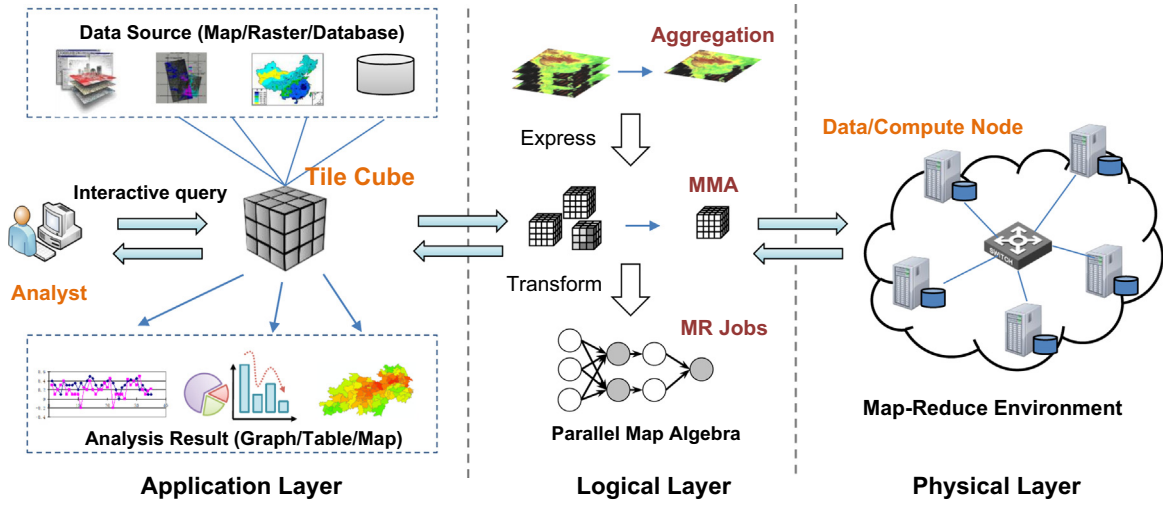

Fig. 3. Overall architecture of Tile Cube model.

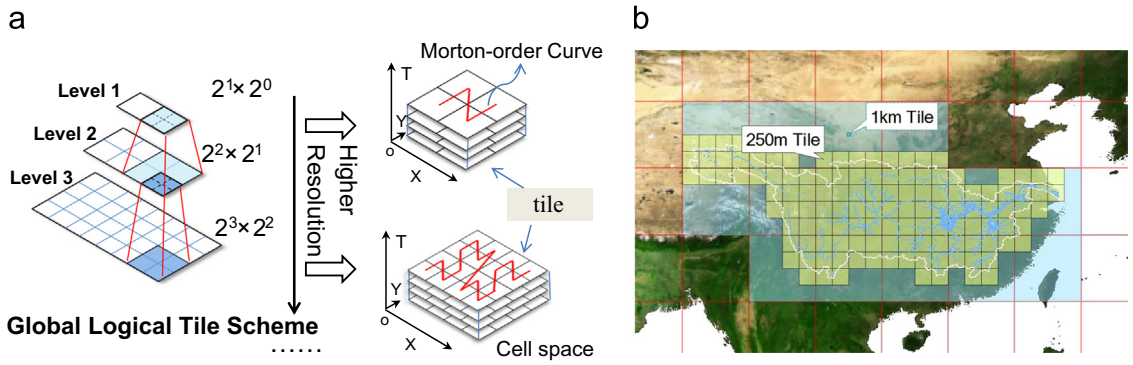

Fig. 4. Global logical tile scheme and Yangtze River Basin represented by tiles.

The tiles belonging to the same geographic cell are called **cell-compatible tiles.** Spatial and spectral analyses of raster are usually performed on the cell-compatible tiles. The assembled tiles can coarsely represent a raster object using $(geom, \{tile\}, tg)$, where $\{tile\}$ is a set of tiles covered by corresponding geographic object *geom* and *tg* is the time tag. Fig. 4(b) shows the Yangtze River Basin represented by tiles in 1 km and 250 m grids separately. In addition to tile, the measure *Representative Object* (**RO**) defined as (p, val, tg) (***p*** *for the pointer to a geographic object*, ***val*** *for the attribute value of the object and* ***tg*** *for time tag*) is introduced to uniformly describe the numeric and spatial measures (*geometry*) at grid level along spatial dimension.

3.1.2. Spatial dimension

Based on the grid, the unique hierarchies or parallel/alternative hierarchies can be built to form spatial dimension (Malinowski and Zimányi, 2008). An example of the spatial dimension with parallel hierarchies (***administrative*** *and* ***valley*** *hierarchies*) is shown in Fig. 5(a), where (▦) represents a tile. There might be multi-relationships between the grid and upper layer, which is referred to as non-strict hierarchy (Pedersen et al., 2001), as shown in Fig. 5(b). The relations between the tile A (B) and the geographic objects are 1:*N* (*N*:*N*) relationships. We denote the part of tile covered by *geom* as **tile**(*geom*) and the objects bounded by tile as **spatial support** of the tile. Thus, the mapping from **tile** (*geom*) to the object can be built via the spatial support, and the object can be described exactly as $(geom, \{tile(geom)\}, tg)$.

3.1.3. Data cube

Tile Cube model perceives time-series raster as a basic data cube, and the spatio-temporal domain of the cube can be described as Eq. (1), where *m* is the measure type; dom_D denotes the domain of cube value along dimension *D*; $\{d_1, \ldots, d_n\}$ are the members in level *l* of *D*. For example, $\{c_1, \ldots, c_n\}$ are the members

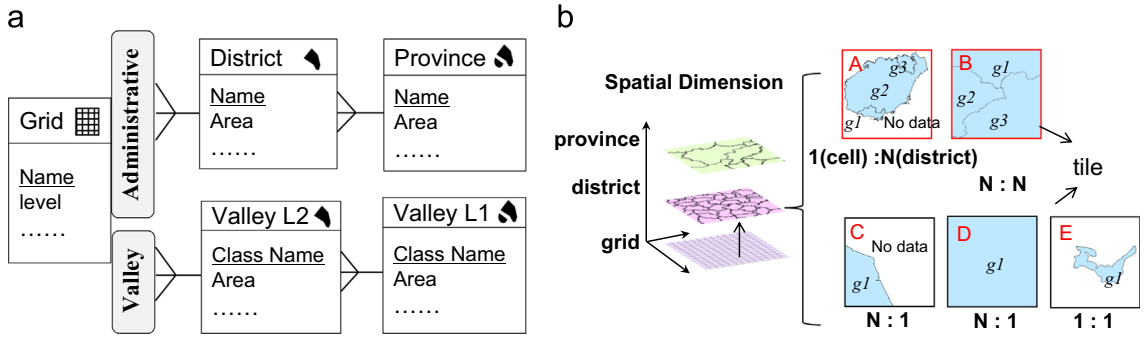

Fig. 5. Spatial dimension of Tile Cube. (a) Non-strict hierarchies based on grid level and (b) Relations between cells and geographic object.

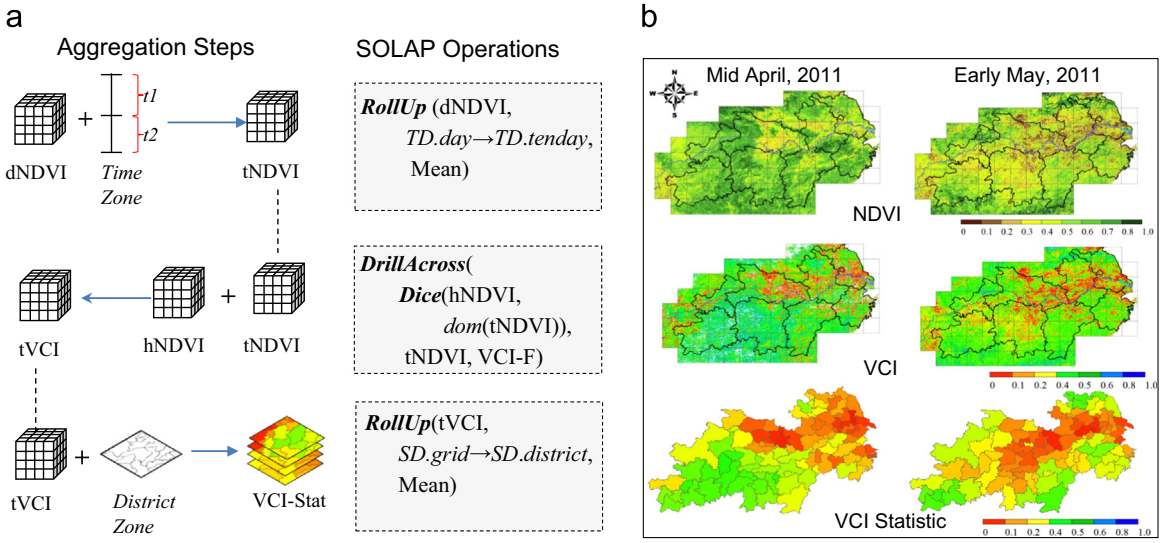

Fig. 6. Example of SOLAP queries of VCI statistics and results for each step.

in spatial dimension (SD); $\{t_1, \ldots, t_n\}$ are the members in time dimension (TD). The data cube can be regarded as a K-V Dataset, and the key of each tile in the data cube is noted as the combination of the measure type and dimension values.

$$\begin{cases} dom = (m, dom_{D_1}, \cdots, dom_{D_n}) \\ dom_D = (l{:}\{d_1, \ldots, d_n\}) \end{cases} \quad (1)$$

3.1.4. Relations between data cubes

If one type of dataset is perceived as a data cube, there would be several cubes in an application-oriented model. The relations between the cubes can be described via a directed graph: $G = \langle V(G), E(G) \rangle$, where $V(G)$ denotes a set of nodes, and each node represents a data cube; $E(G)$ refers to a set of relations: $\{\overset{F}{\rightarrow}\}$. The notation $\overset{F}{\rightarrow}$ explicitly states the relations between cubes, and F refers to the aggregation function. The spatial or non-spatial dimensions can be shared by these cubes.

3.2. Aggregations in Tile Cube model

Because of the storage limitation, it is not practical to pre-compute all the cube measures. Therefore, some of the high-level measures would be computed through online queries and output visually by charts/graphs. There are three types of aggregations in the Tile Cube: aggregation along spatial dimension (AGS), aggregation along non-spatial dimension (AGN) and aggregation between cubes (AGC).

Actually, AGS and AGN are *Spatial* Roll-Up and *Time* Roll-Up operations, respectively, and their formalised SOLAP expression is shown in Eq. (2), where the data cube is aggregated from l_i to upper layer l_j along dimension D using function F.

$$cube\_1 = RollUp(cube, D.l_i \rightarrow D.l_j, F) \quad (2)$$

The AGS from a geographic layer to upper layer already exists in previous SOLAP (Malinowski and Zimányi, 2008), and thus, the AGS discussed here focuses on the aggregation from the grid to upper layer. If the geographic object *geom* in l_j covers a group of cells {*cell*} in grid layer, the raster object ***R*** bounded by *geom* can comprise a group of tiles {*tile*} corresponding to {*cell*}. Thus, the representative object ***RO*** can be obtained using a Zonal function as shown in Eq. (3) if $RO.p \rightarrow R.geom$ and $RO.tg = R.tg$.

$$RO.val = Zonal(F, R.\{tile\}, R.geom) \quad (3)$$

where F is the aggregate function; $R.\{tile\}$ is the input raster and $R.geom$ is an input zone of the Zonal function. If $R.\{tile\}$ is a data cube, this computation can be considered as a cubic Zonal function in MMA.

For AGN, given l_i and l_j are consecutive related levels in non-spatial dimension ***D*** (e.g., *time dimension*) with the relation of $l_i \rightarrow l_j$, the raster objects $r_1, \ldots, r_n$ in level l_i can be aggregated to r_o in level l_j along D if $r_o.geom = r_1.geom = \ldots = r_n.geom$. The value of r_o can be computed with the Local function

$$r_o.tiles = Local(F, \langle r_1.tiles, \ldots, r_n.tiles \rangle) \quad (4)$$

where each tile in raster r_o is computed by a group of cell-compatible tiles independently.

AGC can be considered as ***Drill-Across*** operation and its formalised SOLAP expression is shown in Eq. (5), where the data cube $Cube_o$ in the graph $\boldsymbol{G}$ is aggregated from nodes $cube_1, \ldots, cube_n$ via $\xrightarrow{F}$. This aggregation also can be expressed by the Local function as shown in Eq. (6)

$$cube_o = DrillAccross(cube_1, \ldots, cube_n, F) \tag{5}$$

$$cube_o.tile = Local(F, \langle cube_1.tile, \ldots, cube_n.tile \rangle) \tag{6}$$

where each tile in cube $cube_o$ is computed by a group of tiles in the same position of $cube_1, \ldots, cube_n$ independently.

In drought monitoring, the Vegetation Condition Index (VCI) based on the Normalised Difference Vegetation Index (NDVI) is one of the drought indexes (Kogan, 1995). Fig. 6(a) gives an example of SOLAP queries of VCI statistics based on daily NDVI data to elaborate the aggregations above. In this example, the daily NDVI (**dNDVI**) is composited to a ten-day dataset (**tNDVI**) via aggregation along the time dimension using a **Mean** function (i.e., *Time Roll-Up*). Then, the composited data cube and the historical cube (**hNDVI**) are joined together via aggregation between cubes (i.e., *Drill-Across*) to generate a VCI cube (**tVCI**). Finally, the aggregation along spatial dimension (i.e., *Spatial Roll-Up*) is employed to obtain VCI statistics (**VCI-Stat**) for each district under the support of the District Zone. Fig. 6(b) presents the results in each step within two periods.

4. Implementation of Tile Cube in Map-Reduce

Based on the proposed conceptual model, the implementation of M-R-enabled Tile Cube includes cube data storage and aggregations in the M-R pipeline. In this paper, HBase, a distributed non-relational database that hosts very large tables (made of billions of columns/rows) is employed to store cube data (detailed in Section 4.1). The aggregations on the cube data are implemented on top of Hadoop, a popular open-source M-R framework (detailed in Section 4.2). In the M-R environment including one or more master nodes and several DCNs, the working mechanism of Tile Cube is described in Fig. 7. First, the SOLAP driver transforms SOLAP operations to M-R Jobs. At the same time, the data query domain is phrased to query the condition, and the input parameters of M-R Job are configured automatically. After that, the job is submitted via the Hadoop/HBase client to scheduling queue on the master node. When the job gets ready to run, the task workers on DCNs apply subtasks to master for executing the Map/Reduce phase. Then, the SCAN runs distributed data scanning on DCNs via the **Data Access Interface** (DAI), and each result is consumed as a local input of the Map phase. Finally, the Reduce phase collects the grouped results from Maps, executes the aggregation and writes the finally results into a database. The complexities (*fault tolerance, task assignment and workflow control*) of parallel computing are delivered to Hadoop.

4.1. Storage of cube data in HBase

At a conceptual level, HBase tables are viewed as a sparse set of rows (*row-key is primary key*). Physically, they are stored on a per-column family basis. Column Family (**CF**) comprises several columns. For table cells, the intersection of the row and column coordinates is versioned using a timestamp. The content of the table cell is an

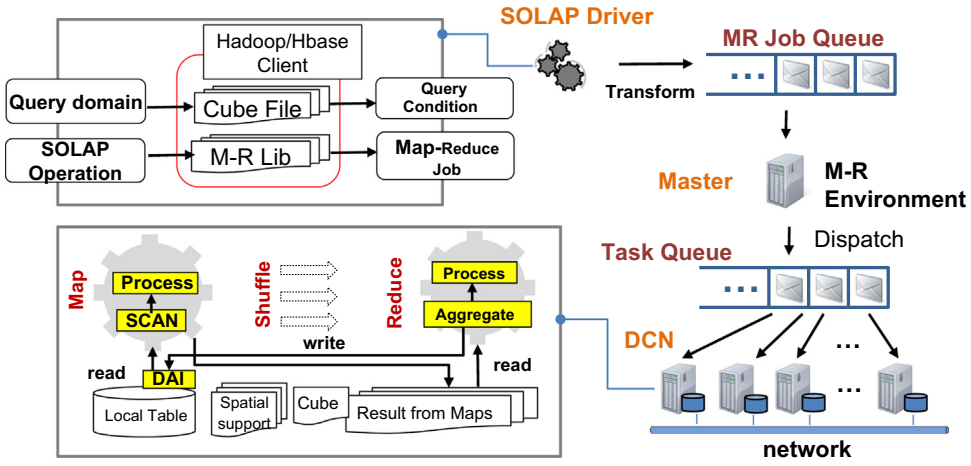

Fig. 7. Working mechanism of aggregations in Tile Cube.

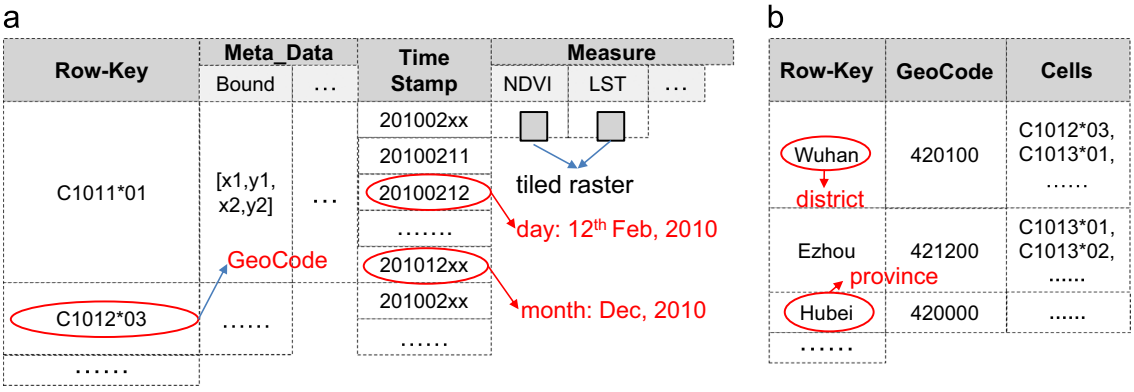

Fig. 8. Design of fact table and index table in HBase.

uninterrupted array of bytes. HBase supports parallel distributed querying and M-R computing on Hadoop clusters.

The fact table for a measure tile (*tile table*) in HBase is designed in Fig. 8(a). Each row stores a group of cell-compatible tiles and the row-key is designed as *geocode* employing the Morton code of the cell. The CF "*Measure*" indicates the tiles of all types, such as NDVI and LST tile. The "*Time_Stamp*" column indicates the time value and the CF "*Meta_Data*" keeps metadata of the cell, such as geographic scope (*bound*). Following this design, the tiles can be stored in the order of cell→time→measure type. We store the tile of all levels in one large table which will be partitioned automatically into multiple blocks over distributed storage regions according to row-key. These designs ensure all the cell-compatible tiles can be stored in the same or closed nodes, thus avoiding the data transfer when performing spatial/spectral analysis within the same geographic domain. For measure RO, the fact table is similar to tile table and the CF "*Measure*" keeps representative values. The spatial supports of tiles are pre-cached in all nodes and each tile can be navigated to its spatial support locally through a row-key. The index table in Fig. 8 (b) describes the relations between cells and the geographic objects in spatial hierarchies. The column "*GeoCode*" stores the geocode of the geographic object. The *cube description file* deployed in the nodes indicates the mapping between the cubes/dimensions and the table structure. Fig. 9(a) provides an example of a "*drought*" cube description file. It shows the members in the "*admin*" hierarchy are stored in the index table "*itable*"; the time information can be obtained from column "*time_stamp*" and the measure "*NDVI*" is mapped to "*Measure: NDVI*", etc. Following that, the spatio-temporal domain can be parsed to the query conditions in row-key and column-values.

$$dom = \langle M.NDVI, SD.grid : \{'Wuhan', 'Ezhou'\}, TD.day : '201103*' \rangle \quad (7)$$

Based on this storage, the cube queries, such as the *Dice* operation, can be implemented in two querying processes: spatial queries and fact queries. Fig. 9(b) shows an example of **Query**: NDVI=Dice(*Cube*, *dom*), where *dom* is given in Eq. (7). First, a spatial query is executed on the index table to obtain the geocode lists of queried regions, "*Wuhan*" and "*Ezhou*". Then the two lists are joined to remove the shared cells. Finally, the fact query maps the joined cell list and other constraints in the domain (e.g. *measure type* and *timestamp*) to query conditions. The built-in scanner of the fact table will take the conditions as query filter, and executes distributed scanning to find matched tiles. In practice, this process is implemented by an M-R job called SCAN.

4.2. Aggregations in the Map-Reduce pipeline

With a Tile Cube, aggregations via MMA functions can be decomposed into multiple traditional map algebra functions based on the granularity of tile. The AGS and aggregate function "**Mean**" are used to elaborate how to decompose the aggregations into multiple parallel tasks in M-R Jobs that take a raster cube as K-V inputs.

For the parallel implementation, the aggregation in AGS can be executed in two steps as shown in Eq. (8), where, $tile_i$ is the *ith* tile in $R.\{tile\}$ ($|R.\{tile\}| = n$), $tile_i(geom)$ denotes the spatial support of $tile_i$. First, the value is aggregated within the tile via Zonal function F_1. Then, the results are collected to feed the aggregation within zone via aggregate function F_2. Fig. 10 gives an example of the VCI average statistic within districts of the "Wuhan" and "Ezhou" districts. In this case, the **Zonal Mean** function is invoked by an NDVI cube within the queried domain *dom* referred to Eq. (7). First, the **Zonal Sum&Count** function (F_1) is performed within each tile. Then, the numeric **Mean** (*Sum*/*Count*) function (F_2) aggregates the measurements within the zone.

$$RO.val = F_2(\cup_{i=1}^{n} F_1(tile_i, tile_i(geom))) \quad (8)$$

The M-R implementation of the processes above is given in Table 1. Given the aggregate geographic zone is Ca, SCAN will run on all Map nodes to query local matched tiles based on the query condition from fact table. In each Map, the time(t), measure type (m) and tile are read from the input $\langle k, v \rangle$, where k is the row-key (geocode of cell), and v denotes the cell values in the row. The **Zonal Sum&Count** function loads the tile and its spatial support tile(Ca) locally and generates the intermediate statistic list G in form of $[\langle geom, (sum, count) \rangle]$, where geom refers to geographic object in Ca**,** and $(sum, count)$ are aggregate results. These results are output to a local disk/memory via the **EMIT** function, and the output Key is combined by m, geom and t via the JointKey function. After that, the Reduce side collects the intermediate results of the same Key from all maps and performs the final aggregation within zone through $f = \sum sum_n / \sum count_n$. The results are finally dumped into HBase via the **EMIT** function. The local Reduce, namely Combine, aggregates the local data via the Sum

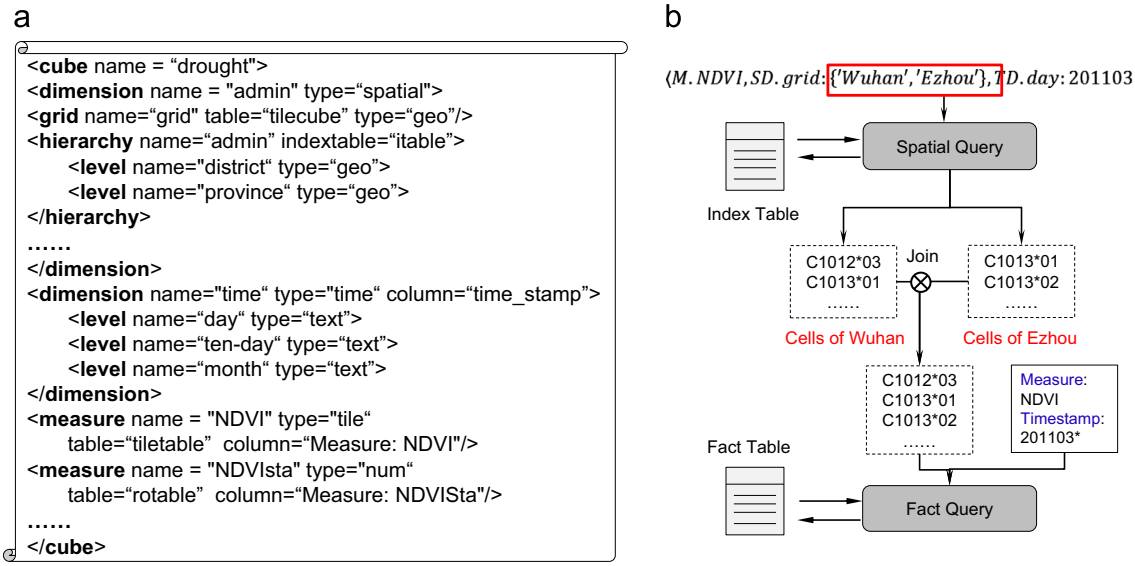

Fig. 9. Example of cube description file and query process in Tile Cube.

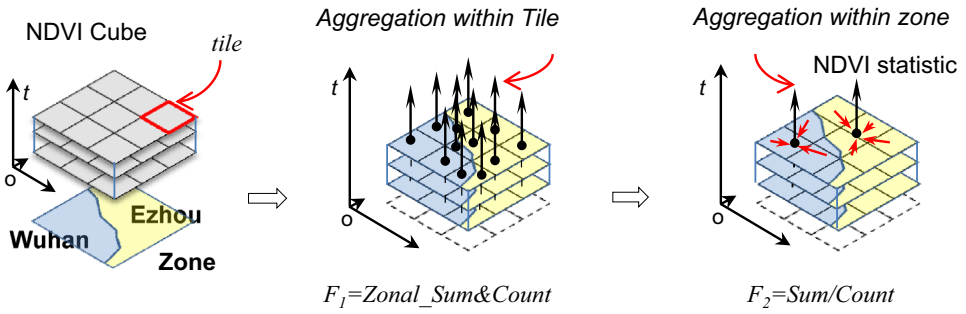

Fig. 10. Example of AGS: VCI statistic.

Table 1
M–R implementation of AGS by mean function.

Algorithm 1 AGS by Mean function in M–R
// *input from table scan with query condition*
// ***k***: *geocode of cell*, ***v***: *cell values of row k*
// **Ca**: *aggregate geographic zone*
MAP(*k*, *v*) // *aggregation within zone*
1 *tile*=**GetCubeTiles**(*v*)
2 *t*=**GetTime**(*v*)
3 *m*=**GetMeasureType**(*v*)
5 let **G** be the list of pairs of *geographic objects* and *its attributes*
6 **G**←**Zonal_Sum&Count**(*tile*, *tile*(Ca))
7 For each <*geom*, (*sum*, *count*)> in **G**
8 *k*=**JointKeys** (*m*, *geom*, *t*)
9 **EMIT** (*k*, (*sum*, *count*))
COMBINE(*k*, list[(*sum*, *count*)]) // *local aggregation*
1 (*sum*, *count*)←**Sum**(list[(*sum*, *count*)])
2 **EMIT** (*k*, (*sum*, *count*))
REDUCE(*k*, list[(*sum*, *count*)]) // *final aggregation within zone*
1 *average* ←**Sum**(list[*sum*])/**Sum**(list[*count*])
2 **EMIT** (*k*, *average*)

Table 2
Time overheads of stand-along and M-R modes in the Tile Cube.

Data size (GB)	Stand-alone (min)				MR1(MR2) (min)			
	NDVI	VCI	VCI-Stat	Overall	NDVI	VCI	VCI-Stat	Overall
1.3	0.7	0.4	0.3	1.4	0.8(0.7)	0.4(0.4)	0.3(0.3)	1.5(1.4)
15.9	11.7	5.3	4.3	21.2	1.4(1.1)	0.9(0.6)	0.5(0.4)	2.5(2.2)
48.4	28.8	19.2	12.0	60.0	1.6(1.3)	1.1(1.0)	0.5(0.5)	3.0(2.7)
142.6	86.8	52.1	34.7	173.5	4.4(3.8)	3.2(2.4)	1.5(1.5)	8.4(7.4)
552.1	316.5	200.3	129.2	646.0	16.1(13.6)	9.3(8.7)	5.5(5.2)	30.9(27.2)

function previously before the remote Reduce, which greatly reduces the network traffics during shuffling.

AGC and AGN are computed by Local functions where each tile is calculated independently by a group of cell-compatible tiles without communications among spatial neighbours. Therefore, they are embarrassingly parallel computations in M-R implementations. The detailed examples and algorithms of AGN and AGC can be found in (http://homegisfly.blogspot.com). The communications between the Map and Reduce phase are based on K-V, and thus, different functions can be assembled by pairs of Map/Reduce phases, such as mathematical (*Add*, *Subtract*, *Multiply*, *Divide*), statistical (*Maximum*, *Minimum*, *Mean*) and comparative (*GreatThan*, *LessThan*, *EqualTo*) functions in different aggregations that form the M-R Job library. Apart from the data locality in the Map/Combine phase, the Locality-Aware Reduce Task Scheduling is also employed in the Tile Cube (Hammoud and Sakr, 2011). This method selects the Map node that emits the largest data size as the Reduce node, thus avoiding unnecessary data transfer.

5. Performance

The distributed environment is built on two versions of Hadoop/HBase (the old version is Hadoop 0.20.2/HBase 0.92.0, the new version is Hadoop YARN 2.2.0/HBase 0.96.0) (Shah et al., 2013)) with default configurations, including eight DCNs and one master node. Each DCN is equipped with 2.4 GHz four dual-core CPUs with 12 GB of RAM in the 1 Gbps network. The prototype implemented by JAVA employs GDAL (http://www.gdal.org/) and MMA (http://code.google.com/p/mdma/) libraries for data operating. As a comparison, PostgreSQL9.3/PostGIS2.1.0 (*relation database*) is selected as a back-end database to construct a stand-alone data warehouse for drought monitoring. PostGIS has provided Map Algebra functions that can be used to implement the aggregations via *pgsql* (PostGIS Raster, 2013). The 250 m MODIS NDVI (MODIS, 2013) collected in April 2011 over China was used. All of the test data were loaded into the PostgreSQL and HBase previously with the tile size $1.25^\circ \times 1.25^\circ$ ($n=7$).

The example in Section 3.2 is employed as the experimental scenario to test the efficiency and scalability of the Tile Cube in the case of data and node growth. The whole process can be implemented by three linked M-R Jobs (NDVI Job, VCI Job and VCI-Stat Job). The two M-R implementations based on the different versions of Hadoop/HBase are called the MR1 mode (*old version*) and MR2 mode (*new version*). Groups of data with the size 1.3 GB (*partially obtained within one ten-day*), 15.9 GB (*one ten-day*), 48.4 GB (*one month*), 142.6 GB (*three months*) and 552.1 GB (*one year*) were obtained and fed for processing into the stand-alone mode and MR1 mode and MR2 mode separately. The time overheads are reported in Table 2, and the speedups of two M-R modes are shown in Fig. 11(a).

Table 2 shows the stand-alone mode keeps growth linear when the data size increases. Because of the delay of Hadoop, the performance of all the M-R Jobs can only achieve the performance of the stand-alone mode when processing one ten-day dataset. However, when processing the data for one year, the speedups of MR1 (MR2) are improved considerably. The results show the acceleration and scalability of M-R modes along with data growth. Because of the efficient memory utility and reasonable scheduling in YARN, MR2 performs better than MR1 when data size increases. When processing 15.9 GB and 48.4 GB data, the time overheads of the M-R mode along with increased nodes (according to the sequences 1, 2, 4, 6, 8) are recorded separately. Fig. 11(b) shows that the speedups of the two cases can both achieve linear acceleration (the storage is scaled with the number of nodes). The speedup in the case of 15.9 GB data can grow slower than the case of 48.4 GB data because the computer resources have not been fully loaded when the node number increases. This test shows the scalability of the M-R mode along with node growth. Moreover, it is evident that YARN has better performance than the old version when the node number increases as well.

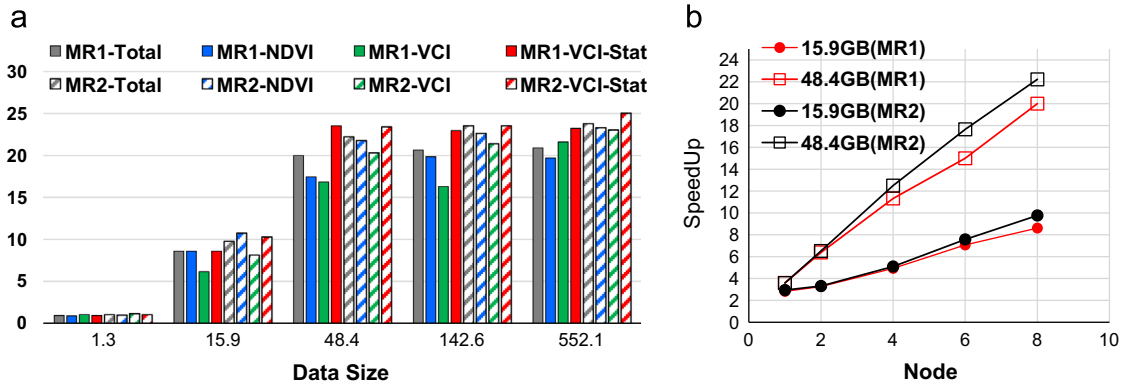

Fig. 11. Speedups of M-R modes with data/node growth. (a) Speedup changes with increased data size and (b) Speedup changes with increased notes.

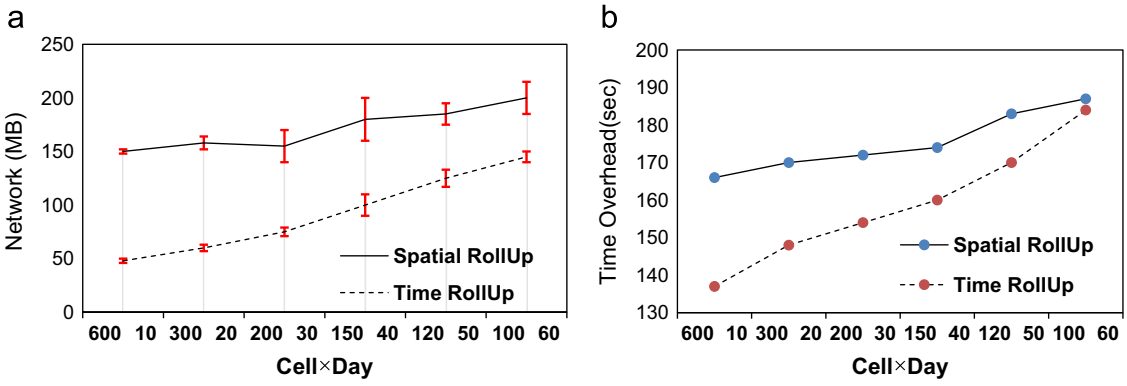

Fig. 12. Impact of data size on performance of SOLAP in Tile Cube. (a) Impact of data size on network and (b) Impact of data size on execution time.

Data size and tile size are two important performance influence factors of M-R modes. The tests above have demonstrated the impact of data increasing on performance. There are two ways to increase the amount of cube data: increasing along the spatial dimension (called *space mode*) and increasing along the time dimension (called *time mode*). To investigate the impact of these two cases, the network/time overhead of the ten-day NDVI (*Time Roll-Up*) composition and NDVI statistics (*Spatial Roll-Up*) are tested when the data are set to 6 groups (*cell* × *time*), and each group contains 6000 tiles. All the tests in the MR2 mode are run for 5 times and the measurements are averaged as shown in Fig. 12. The results show that the time overheads of the two cases both increase when reducing the spatial range and increasing the composite time. As mentioned before, the cell-compatible tiles are stored closely to follow the data locality principle. Therefore, increasing the spatial range helps improve the acceleration. However, in the case of long time series and small spatial range, the engaged nodes will be heavily loaded. Thus, the new tasks will be forwarded to idle nodes, which produces loads of network traffic. Therefore, it is essential to balance the proportion between data locality and distributed computation for optimisation.

The tile size is related to the granularity of the M-R Job. The VCI statistic of the ten-day NDVI sequence is employed to test the time overhead in the cases of three tile sizes ($0.625^\circ \times 0.625^\circ$, $1.25^\circ \times 1.25^\circ$, $2.5^\circ \times 2.5^\circ$). All the tests are run for 5 times and the measurements are averaged as shown in Fig. 13. The results show that the overheads of *Spatial* Roll-Up and Drill-Across both reduce when the tile size increases. This is because that the partition with small size causes a large amount of Map tasks that will lead to heavy loads from thread management and task scheduling (if the quad-tree partitioning is executed for one time, the number of Map tasks will increase by four times theoretically). On the

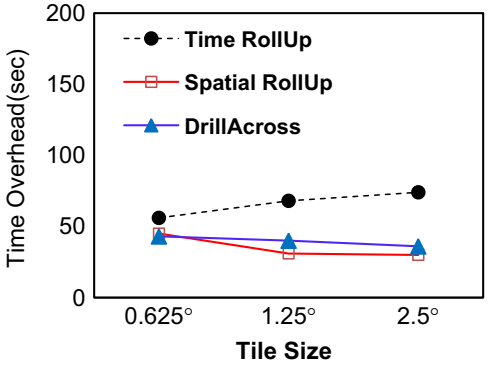

Fig. 13. Impact of tile size on performance of SOLAP in Tile Cube.

contrary, the performance of *Time* Roll-Up is improved when tile size increases. The multi-day composition in the Reduce phase has more computational complexity, and thus, the task with small size helps improve the parallelism. This test shows the task with more computational complexity can benefit from small tile size and obtain better acceleration.

Apart from the factors mentioned above, the network speed and M-R phases overlapping also affect the performance of aggregations in the Tile Cube (Lin et al., 2013). Therefore, balancing these factors in a specific application is an important research emphasis.

6. Related work

Within the past few years, M-R-based systems have been adopted for large-scale, interactive data querying (e.g., OLAP).

In 2009, the Apache Hive is proposed for providing data summarisation, query, and analysis as a typical data warehouse built on top of Hadoop (Hive, 2013). Instead of building a generic data warehouse as Hive does, Cheetah is designed to allow various custom features and optimisations ranging from schema design and query language (Chen, 2010). In order to reduce the interactive query latency by Hadoop, some practitioners have begun to build low-latency OLAP with HBase (Lehene, 2012; HBase-Lattice, 2014). Abelló has also explored the possibility of using Bigtable to store historical data and M-R as an agile mechanism to deploy cubes in ad-hoc Data Marts (Abelló et al., 2011). Moreover, a few researchers have begun to address the efficient issues on M-R approaches of cube aggregation. For example, Lee proposed an M-R-based algorithm for top-down OLAP cube computation, which reduces the number of data scans by pipelining the processing (Lee et al., 2012). In particular, Nandi addressed the parallelism problem of holistic measures of cube materialisation over M-R (Nandi et al., 2011). Unfortunately, these works were not designed for spatial OLAP, and thus, the spatial dimension, spatial measure and map algebra functions are not supported.

The aggregations in field/raster SOLAP are usually implemented by MA/MMA. In recent years, some high-performance computing technologies have been used to extend MA/MMA for improving the throughput and efficiency. For example, Shrestha has implemented parallel temporal map algebra using MatlabMPI for vegetation index composition (Shrestha et al., 2006). The GeoRaster model in the Oracle Database is being presently extended by map algebra in a parallel model (Xie et al., 2012). The parallel raster database Rasdaman has provided SQL-liked operations but based on array algebra (Rasdaman database, 2013). However, the scalability and fault tolerance of these methods have been limited by the traditional high-performance computing paradigm in regard to large-scale data. Raster has natural data parallelism and lots of researchers aimed at employing M-R to accelerate remotely sensed data handling. A 1D ray-tracing algorithm was extended by M-R, which effectively decreases computational effort and improves accuracy (Mohammadzaheri et al., 2013). The real-time gridding of AIRS data implemented over M-R shows its considerable efficiency (Golpayegani and Halem, 2009). Nevertheless, the literature on M-R-enabled MA/MMA available in the form of academic publications is limited. Given the above, our work takes full advantage of the power of both M-R and SOLAP to improve the throughput and scalability of remotely sensed data aggregation.

7. Conclusion

SOLAP is considered a useful approach for exploiting long-time spatial information. However, the remotely sensed data aggregation in SOLAP faces large data challenges, which limits the usage of this technology. Data-intensive scientific computing, which is considered as a "*fourth paradigm*" (Hey et al., 2009), has great potential to address such challenges using distributed computing paradigm, such as Map-Reduce. This research combines the analysis ability of SOLAP and the computing power of Map-Reduce, and thus, the long time-series, wide-range and multi-view queries on remotely sensed data can be responded to in a short time. The model has been used in the Water Information Centre, Ministry of Water Resources, China, for drought monitoring.

To make full use of M-R in SOLAP, several influencing factors, such as network traffic patterns and computational complexity, should be considered in a balanced way for the specific aggregation. In addition, the implementation of the Multi-Dimensional Expressions (MDX) query language based on the Tile Cube is also listed in our future work to support more SOLAP operations and simple data mining.

References

Abelló, A., Ferrarons, J., Romero, O., 2011. Building cubes with MapReduce. In: Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, pp. 17–24.

Ahmed, T.O., Miquel, M., 2005. Multidimensional structures dedicated to continuous spatiotemporal phenomena. In: Proceedings of the British National Conference on Databases. pp. 29–40.

Bédard, Y., Rivest, S., Proulx, M., 2007. Spatial on-line analytical processing (SOLAP): concepts, architectures, and solutions from a geomatics engineering perspective, Data Warehouses and OLAP: Concepts, Architectures and Solutions. IRM Press, London, UK, pp. 298–319.

Bimonte, S., Myoung-Ah, K., 2010b. Towards a model for the multidimensional analysis of field data. In: Proceedings of the 14th east European Conference on Advances in Databases and Information Systems, pp. 58–72.

Bimonte, S., Tchounikine, A., Miquel, M., Pinet, F., 2010a. When spatial analysis meets OLAP: multidimensional model and operators. Int. J. Data Warehous. Min. 6 (4), 33–60.

Chen S., 2010. Cheetah: A high performance, custom data warehouse on top of MapReduce. In: The 36th International Conference on Very Large Data Bases, Singapore, pp. 1459-1468.

Dean, J., Ghemawat, S., 2004. MapReduce: simplified data processing on large clusters. In: Proceedings of the 6th Symposium on Operating Systems Design and Implementation, pp. 137–150.

Golpayegani, N., Halem, M., 2009. Cloud computing for satellite data processing on high end compute clusters. In: Proceedings of the IEEE International Conference on Cloud Computing, Bangalore, pp. 88–92.

Gómez, L., Gómez, S., Vaisman, A., 2010b. A generic data model and query language for spatiotemporal OLAP cube analysis. In: Proceedings of the 15th International Conference on Extending Database Technology, Berlin, Germany, pp. 300–311.

Gómez, L., Vaisman, A., Zimányi, E., 2010a. Physical design and implementation of spatial data warehouses supporting continuous fields. In: Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery, Italy, pp. 25–29.

Hammoud, M., Sakr, M.F., 2011. Locality-aware reduce task scheduling for MapReduce. In: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, Washington, pp. 570–576.

Hey, T., Tansley, S., Tolle, K., 2009. The fourth paradigm: data-intensive scientific discovery. Microsoft Corp., 1–5.

Hive, 2013. Apache Software Foundation. ⟨http://hive.apache.org/⟩ (accessed 15.11.13.).

Kogan, F.N., 1995. Application of vegetation index and brightness temperature for drought detection. Adv. Space Res. 11, 91–100.

Lee, C.A., Gasster, S.D., Plaza, A., Chang, C., Huang, B., 2011. Recent developments in high performance computing for remote sensing: a review. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 4 (3), 508–527.

Lee, S., Jinho, J., Moon, Y., Lee, W., 2012. Efficient distributed parallel top-down computation of rolap data cube using MapReduce. In: Proceedings of the 14th International Conference on Data Warehousing and Knowledge Discovery, pp. 168–179.

Lehene C., 2012. Low Latency "OLAP" with HBase. ⟨http://www.cloudera.com/content/cloudera/en/resources/library/hbasecon/video-hbasecon-2012-low-latency-olap-with-hbase.html⟩ (accessed 02.06.14).

Lin, M., Zhang, L., Wierman, A., Tan, J., 2013. Joint optimization of overlapping phases in MapReduce. Perform. Eval. 70 (10), 720–735.

Longley, P.A., Goodchild, M.F., Maguire, D.J., Rhind, D.W., 2001. Geographic Information Systems and Science. John Wiley and Sons, New York p. 74.

Lopez, I.F.V., Snodgrass, R.T., Moon, B., 2005. Spatiotemporal aggregate computation: a survey. IEEE Trans. Knowl. Data Eng. 17 (2), 271–286.

Malinowski, E., Zimányi, E., 2008. Advanced Data Warehouse Design. Springer, Berlin p. 4 (76, 104, 147).

McHugh R., 2008. Intégration De La Structure Matricielle Dans Les Cubes Spatiaux. Thèse (M.Sc.)-Université Laval.

Mennis, J., 2010. Multidimensional map algebra: design and implementation of a spatio-temporal GIS processing language. Trans. GIS 14 (1), 1–21.

Mennis, J., Viger, R., Tomlin, C.D., 2005. Cubic map algebra functions for spatio-temporal analysis. Cartogr. Geogr. Inf. Sci. 32 (1), 17–32.

MODIS, 2013. National Aeronautics and Space Administration. ⟨https://lpdaac.usgs.gov/products/modis_overview⟩ (accessed 15.11.13.).

Mohammadzaheri, A., Sadeghi, H., Hosseini, S.K., Navazandeh, M., 2013. DISRAY: a distributed ray tracing by map-reduce. Comput. Geosci. 52, 453–458.

Nandi, A., Yu, C., Bohannon, P., Ramakrishnan, R., 2011. Distributed cube materialization on holistic measures. In: Proceedings of the IEEE 27th International Conference on Data Engineering, pp. 183–194.

Pedersen, T.B., Jensen, C.S., Dyreson, E.C., 2001. A foundation for capturing and querying complex multidimensional data. Inf. Syst. 26 (2001), 383–423.

PostGIS Raster, 2013. PostGIS, Open Source Geospatial Foundation. ⟨http://postgis.net/docs/RT_reference.html⟩ (accessed 15.11.13.).

Sample, J.T., Loup, E., 2010. Tile-based geospatial information systems, Principles and Practices. Springer, New York, pp. 7–9.

Shah, H., Seth, S., Saha, B., Curino, C., et al., 2013. Apache Hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing, Santa Clara, California, pp. 1–16.

Shrestha, B., O'Hara, C.G., Younan, N.H., 2006. Strategies for alternate approaches for vegetation indices compositing using parallel temporal map algebra.

In: Proceedings of the ASPRS 2006 Annual Conference, Reno, Nevada, pp. 1–11.
Tomlin, C.D., 1991. Cartographic modelling. Geographic information systems: Principles and applications. Longman Scientific & Technical, Essex, U.K., pp. 361–374.
Vaisman, A., Zimányi, E., 2009. A multidimensional model representing continuous fields in spatial data warehouses. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, USA, pp. 168–177.
White, T., 2009. Hadoop: The Definitive Guide. O'Reilly Media, Sebastopol, pp. 23–26.
Xie Q., Zhang Z., Ravada S., 2012. In-Database raster analytics: Map algebra and parallel processing in Oracle Spatial GeoRaster. In: XXII Congress of the International Society for Photogrammetry and Remote Sensing, Melbourne, pp. 91-96.
Zhang, J., 2006. Spatio-temporal aggregates over streaming geospatial image data. In: Proceedings of the International Conference on Extending Database Technology 2006 Workshops, pp. 32–43.
Zhang, J., Yang, W., Sun J., Lv, Y., 2010. GPU-Accelerated parallel algorithms for map algebra. In: Proceedings of the 2nd Conference on Environmental Science and Information Application Technology, Wuhan, China, pp. 882–885.